**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**



**Лабораторна робота №2**

з дисципліни

«Технології розподілених систем та паралельних обчислень»

**Виконав:**

студент групи КН-308

Гецянин Дмитро

**Викладач:**

Мочурад Л.І.

Львів – 2020р.

**Лабораторна робота №2**

**Тема: Розподіл роботи між потоками засобами OpenMP**

**Мета:** Вивчити розподіл роботи між потоками засобами OpenMP і навчитися їх застосовувати.

**Варіант №3**

**Завдання.**

Створити програму яка повинна реалізувати наступні дії:

1. Створити квадратні матриці А та В розміром n\*n, елементи яких заповнюються

довільними числами, n задає кількість рядків і кількість потоків, які виконуватимуть

паралельну область програми. Змінні можуть задаватися в програмному коді або

вводитися з клавіатури.

1. У кожній матриці окремо обчислити паралельним способом завдання відповідно до

свого варіанту. Розподіл ітерацій між потоками виконати за допомогою директиви for з

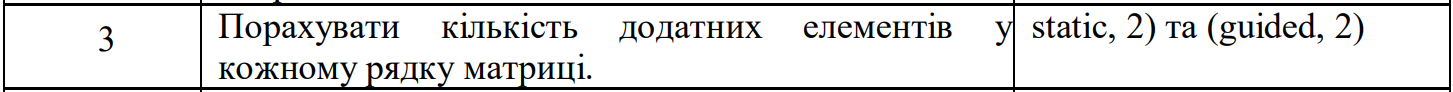
використанням опції schedule. Для кожної з матриць використати різні значення

параметру type та розміру блоку – chunk. Результати обробки записати в масиви С та D.

1. Вивести результати обробки матриць паралельним способом, вказавши при виводі

розподіл ітерацій по потоках для різних значень опції schedule. Порівняти результати,

отримані при різних значеннях параметру type та розміру блоку – chunk.



**Програмний код**

#include "pch.h"

#include <iostream>

#include <thread>

#include <time.h>

#include <omp.h>

**using** **namespace** std;

//to show matrix

**void** **show**(**int** \*\*mat, **int** n, **int** m) {

cout << "**\n**";

**for** (**int** i = **0**; i < n; ++i) {

**for** (**int** j = **0**; j < m; ++j) {

cout << mat[i][j] << " ";

}

cout << '\n';

}

cout << "**\n**";

}

//to show array

**void** **show**(**int**\* arr, **int** size) {

cout << "**\n**";

**for** (**int** i = **0**; i < size; ++i) {

cout << arr[i] << " ";

}

cout << "**\n**";

}

//count the positive numbers in the matrix in each row

**int** **count\_a**(**int** \*\*matrix, **int** \*result, **int** size) {

**int** i, j;

#pragma omp parallel shared(matrix,result) private(i,j)

{

#pragma omp for schedule(static, 2)

**for** (i = **0**; i < size; i++) {

**int** count = **0**;

**for** (j = **0**; j < size; j++) {

**if** (matrix[i][j] < **0**)

count++;

}

result[i] = count;

printf("Iteration %d Thread %d**\n**", i, omp\_get\_thread\_num());

}

}

**return** **0**;

}

**int** **count\_b**(**int** \*\*matrix, **int** \*result, **int** size) {

**int** i;

**int** j;

#pragma omp parallel shared(matrix,result) private(i,j)

{

#pragma omp for schedule(guided, 2)

**for** (i = **0**; i < size; i++) {

**int** count = **0**;

**for** (j = **0**; j < size; j++) {

**if** (matrix[i][j] < **0**)

count++;

}

result[i] = count;

printf("Iteration %d Thread %d**\n**", i, omp\_get\_thread\_num());

}

}

**return** **0**;

}

**int** **main**()

{

**int** size\_n;

**int**\*\* matrix\_a;

**int**\*\* matrix\_b;

**int**\* result\_c;

**int**\* result\_d;

cout << "Input n:**\n** n = ";

cin >> size\_n;

matrix\_a = **new** **int**\*[size\_n];

matrix\_b = **new** **int**\*[size\_n];

result\_c = **new** **int**[size\_n];

result\_d = **new** **int**[size\_n];

**for** (**int** i = **0**; i < size\_n; ++i) {

matrix\_a[i] = **new** **int**[size\_n];

matrix\_b[i] = **new** **int**[size\_n];

}

//fills the matrix A with random numbers in range (-100; 100)

**for** (**int** i = **0**; i < size\_n; ++i) {

**for** (**int** j = **0**; j < size\_n; ++j) {

matrix\_a[i][j] = (rand() % **200**) - **100**;

}

}

show(matrix\_a, size\_n, size\_n);

//fills the matrix B with random numbers in range (-100; 100)

**for** (**int** i = **0**; i < size\_n; ++i) {

**for** (**int** j = **0**; j < size\_n; ++j) {

matrix\_b[i][j] = (rand() % **200**) - **100**;

}

}

show(matrix\_b, size\_n, size\_n);

omp\_set\_num\_threads(**4**);

count\_a(matrix\_a, result\_c, size\_n);

show(result\_c, size\_n);

omp\_set\_num\_threads(**8**);

count\_b(matrix\_b, result\_d, size\_n);

show(result\_d, size\_n);

//free up memory

**for** (**int** i = **0**; i < size\_n; ++i) {

**delete**[] matrix\_a[i];

**delete**[] matrix\_b[i];

}

**delete**[] matrix\_a;

**delete**[] matrix\_b;

**delete**[] result\_c;

**delete**[] result\_d;

**return** **0**;

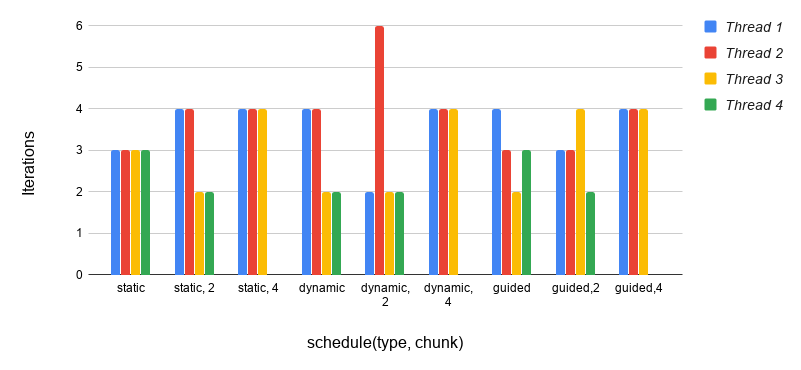
}

**Результати виконання програмного коду**

Результати виконання прикладу з різними типами розподілу ітерацій приведені в таблиці нижче. Стовпці відповідають різним типам розподілів, а рядки – номеру ітерації. У таблиці вказані номера потоків, що виконували відповідну ітерацію. У всіх випадках для виконання паралельного циклу використовувалися 4 потоки. Для динамічних способів розподілу ітерацій (dynamic, guided) конкретне ділення між потоками може відрізнятися від запуску до запуску.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| schedule iterations | static | static, 2 | static, 4 | dynamic | dynamic, 2 | dynamic, 4 | guided | guided,2 | guided,4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 0 | 2 | 1 | 0 | 1 | 1 | 0 |
| 4 | 1 | 2 | 1 | 0 | 2 | 1 | 1 | 1 | 1 |
| 5 | 1 | 2 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| 6 | 2 | 3 | 1 | 0 | 3 | 1 | 2 | 2 | 1 |
| 7 | 2 | 3 | 1 | 3 | 3 | 1 | 2 | 2 | 1 |
| 8 | 2 | 0 | 2 | 1 | 1 | 2 | 3 | 3 | 2 |
| 9 | 3 | 0 | 2 | 2 | 1 | 2 | 3 | 3 | 2 |
| 10 | 3 | 1 | 2 | 1 | 1 | 2 | 3 | 2 | 2 |
| 11 | 3 | 1 | 2 | 0 | 1 | 2 | 0 | 2 | 2 |

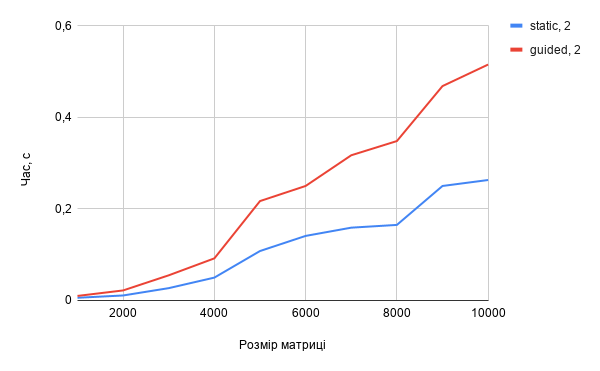
*Таблиця 1. Розподіл ітерацій по потокам в залежності від параметрів schedule*



*Графік 1. Розподіл імерацій між потоками в залежності від параметрів schedule*

|  |  |  |  |
| --- | --- | --- | --- |
| **Розмір матриці** | **Час, сек. (static, 2)** | **Час, сек. (guided, 2)** | **Відносне**  **прискорення** |
| 1000 | 0,006 | 0,010 | 1,667 |
| 2000 | 0,011 | 0,022 | 2,000 |
| 3000 | 0,027 | 0,055 | 2,037 |
| 4000 | 0,050 | 0,092 | 1,840 |
| 5000 | 0,108 | 0,217 | 2,009 |
| 6000 | 0,141 | 0,250 | 1,773 |
| 7000 | 0,159 | 0,317 | 1,994 |
| 8000 | 0,165 | 0,348 | 2,109 |
| 9000 | 0,250 | 0,468 | 1,872 |
| 10000 | 0,263 | 0,515 | 1,958 |

*Таблиця 2. Час обчислення при різних параметрах type та chunk, та їх відносне прискорення (static, 1)/(dynamic, 4)*



*Графік 2. Залежність часу обчислення від розміру матриці при різних параметрах type та chunk*

**Висновок.** У даній лабораторні роботі я навчився виконувати розподіл між потоками за допомогою бібліотеки OpenMP опції schedule та її параметрів type та chunk. Зрозумів принципи роботи цих параметрів та різницю між ними.